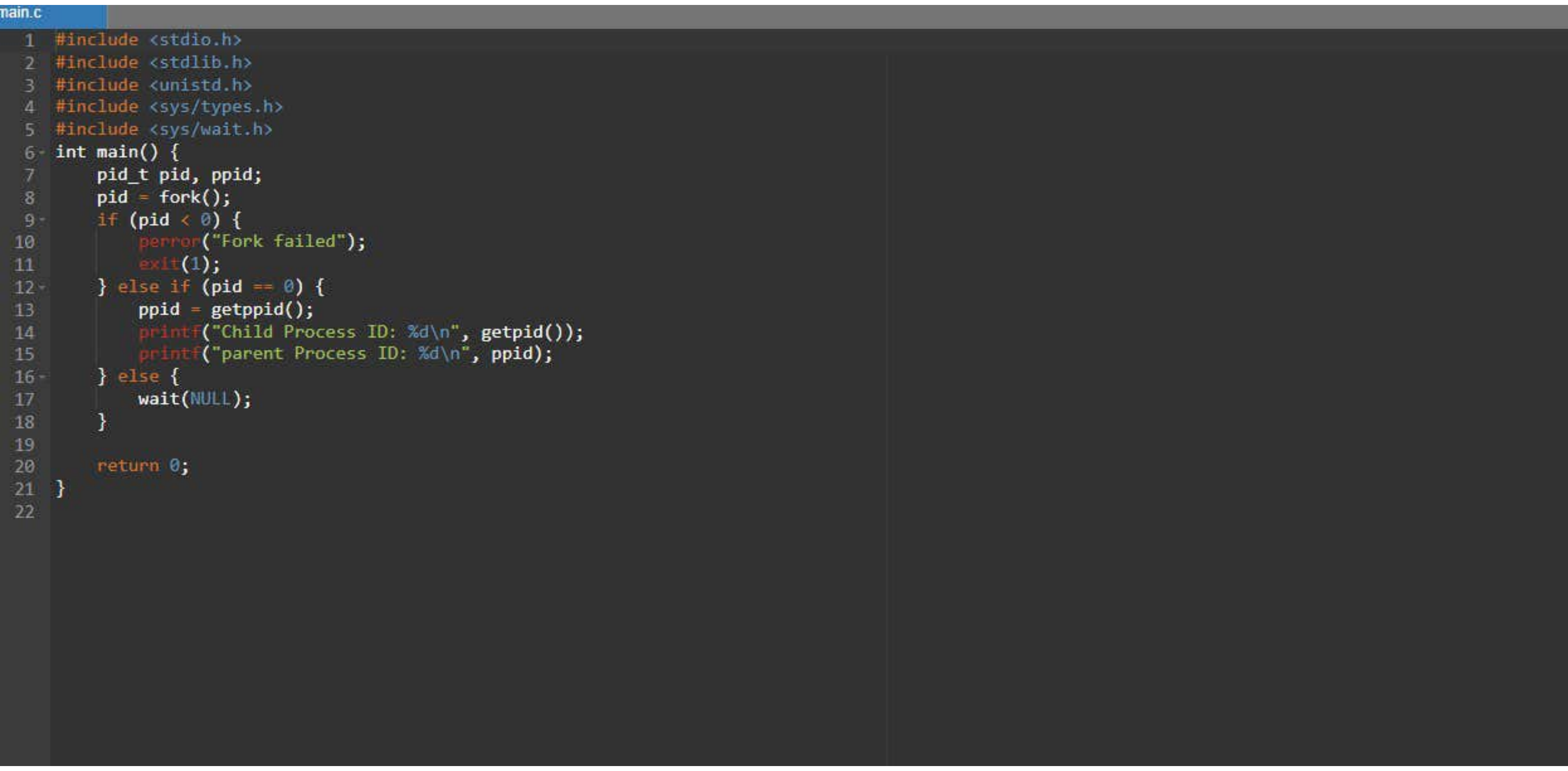
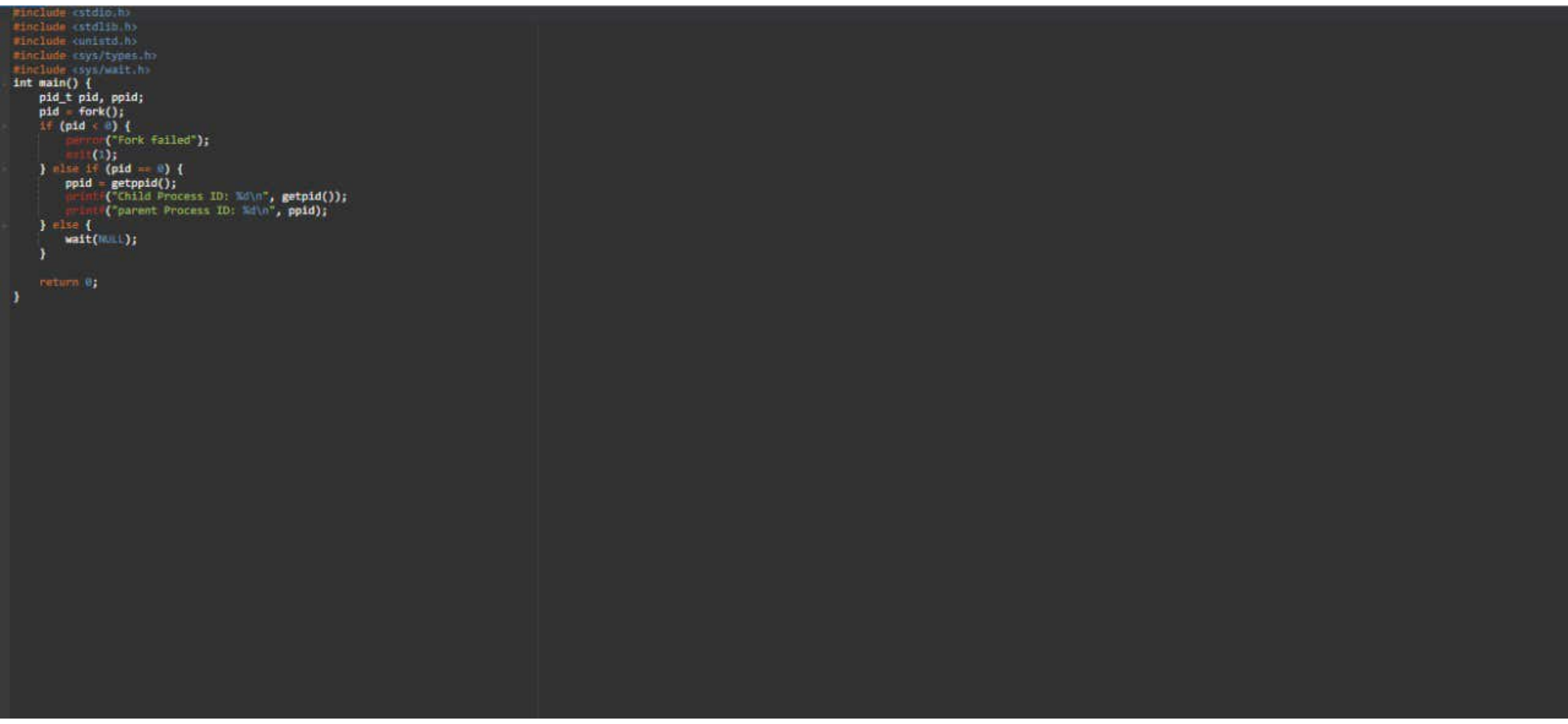
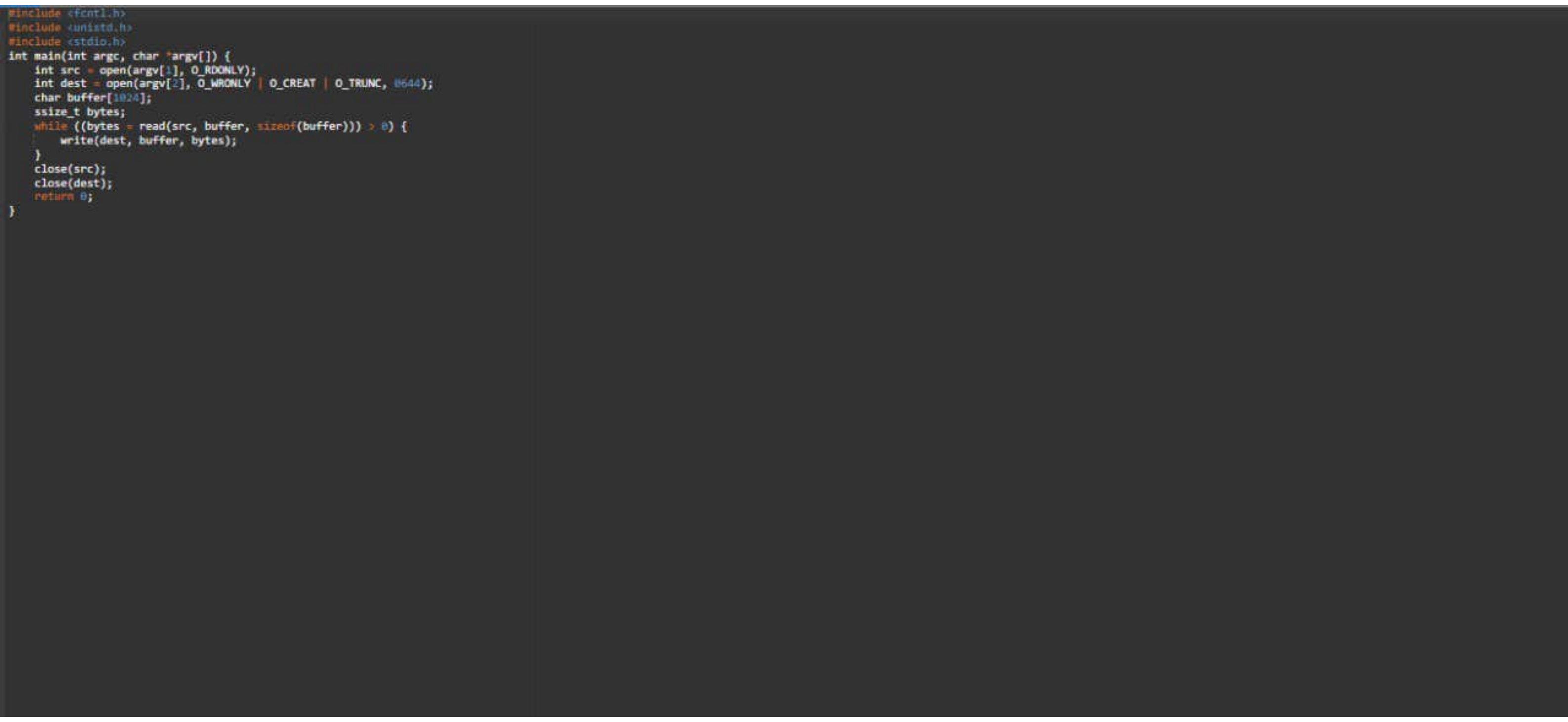


main.c

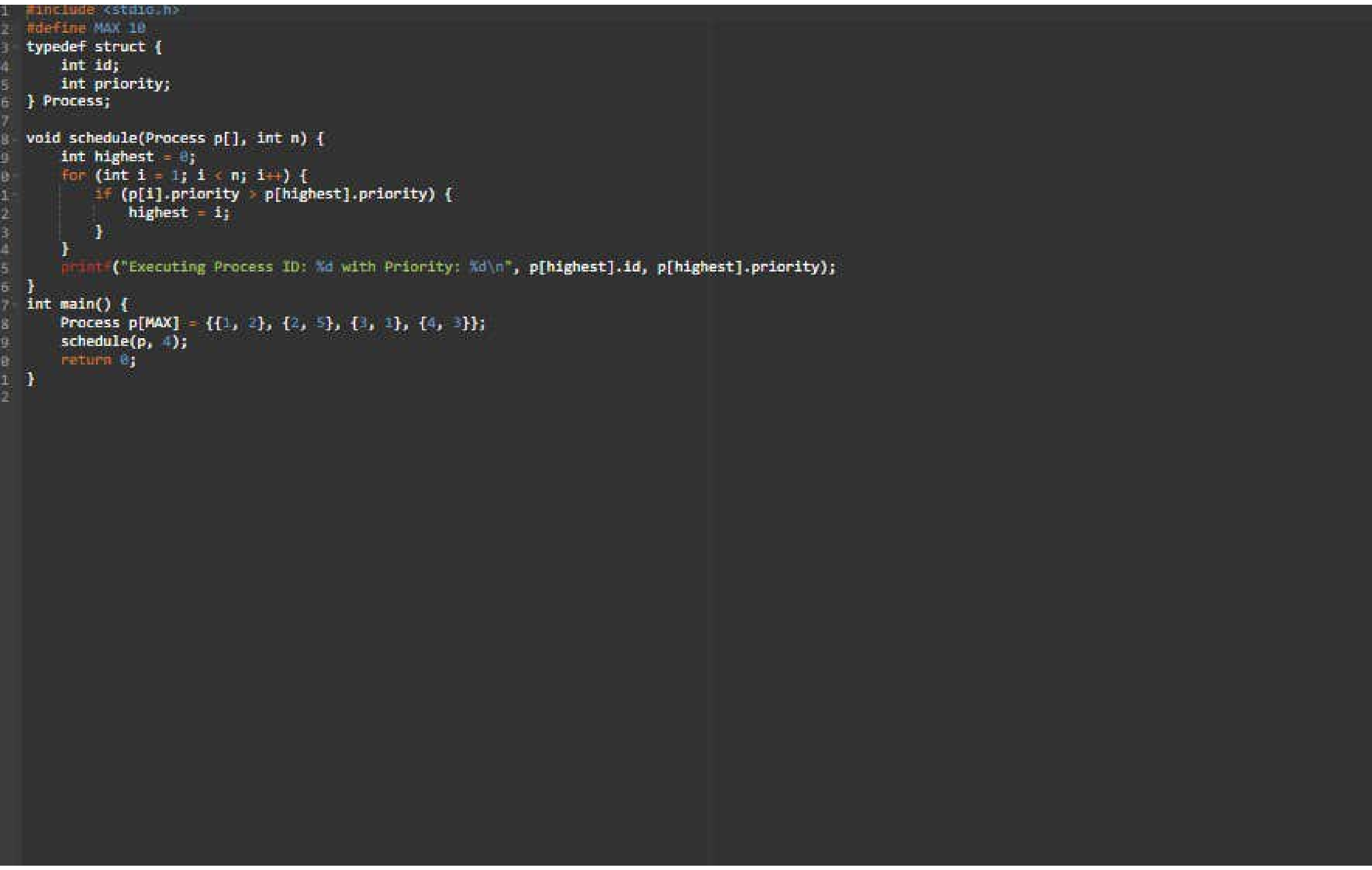
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid, ppid;
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        ppid = getppid();
        printf("Child Process ID: %d\n", getpid());
        printf("parent Process ID: %d\n", ppid);
    } else {
        wait(NULL);
    }

    return 0;
}
```



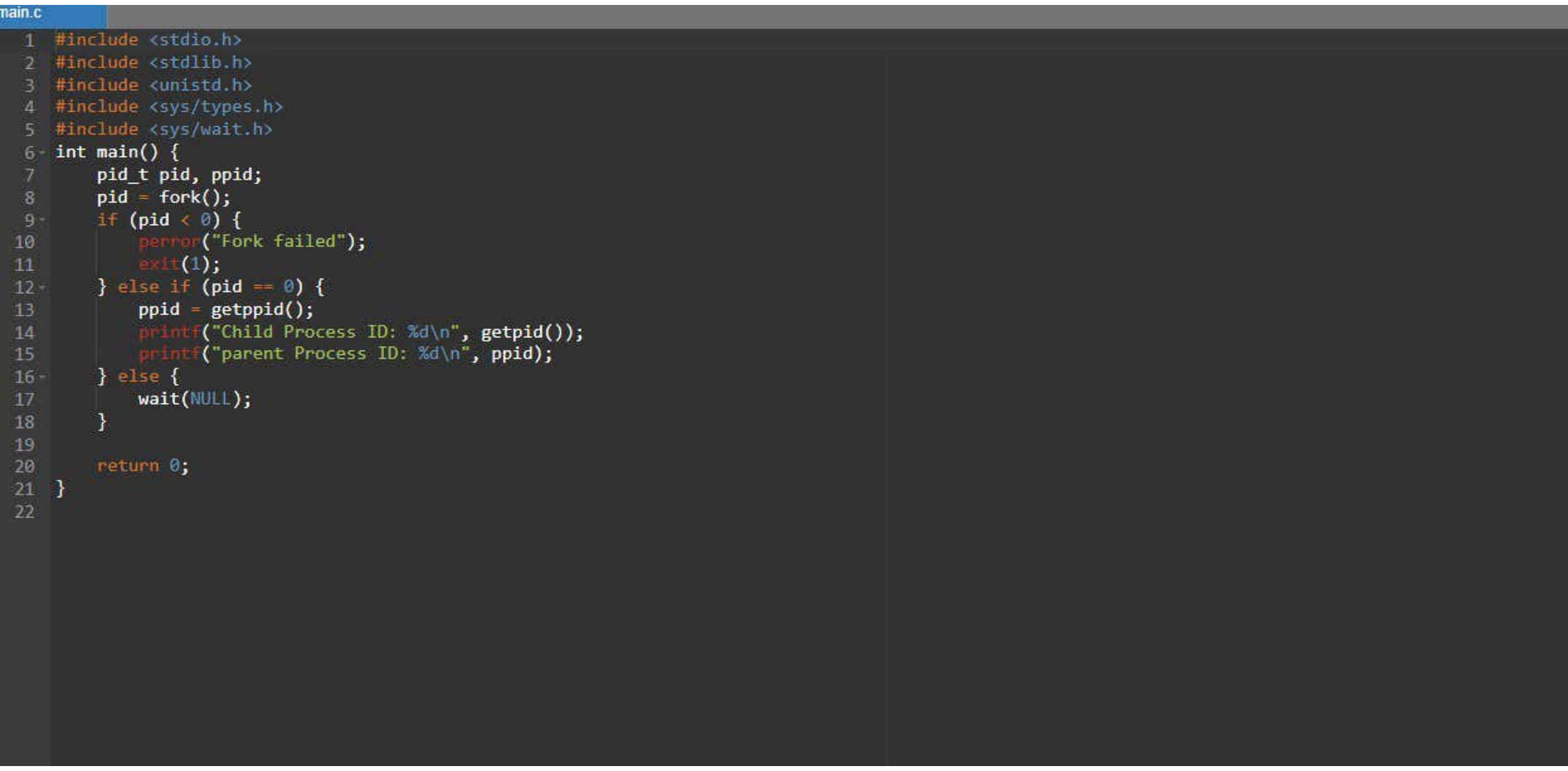
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid, ppid;
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        ppid = getppid();
        printf("Child Process ID: %d\n", getpid());
        printf("parent Process ID: %d\n", ppid);
    } else {
        wait(NULL);
    }

    return 0;
}
```



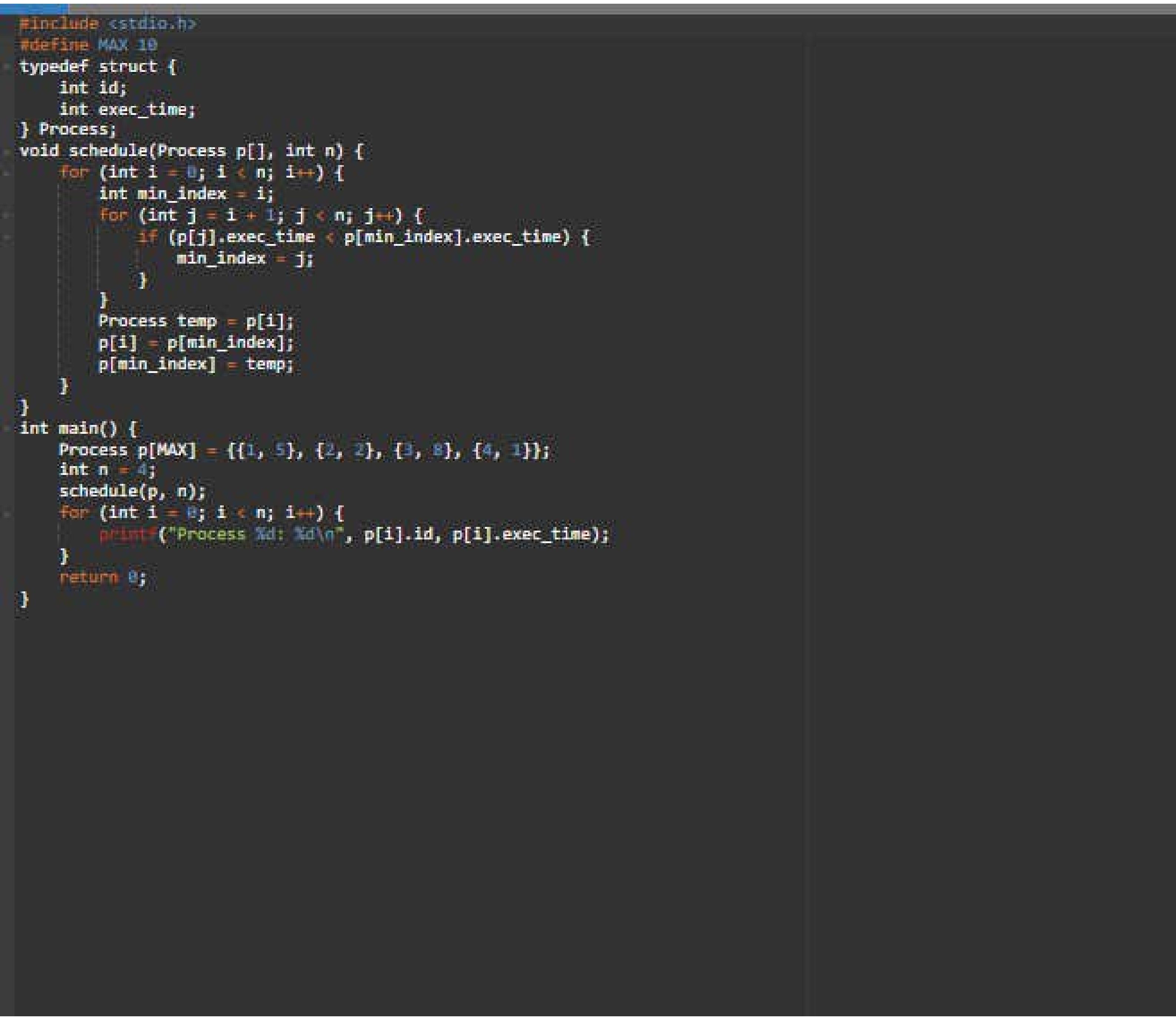
```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    int src = open(argv[1], O_RDONLY);
    int dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    char buffer[1024];
    ssize_t bytes;
    while ((bytes = read(src, buffer, sizeof(buffer))) > 0) {
        write(dest, buffer, bytes);
    }
    close(src);
    close(dest);
    return 0;
}
```



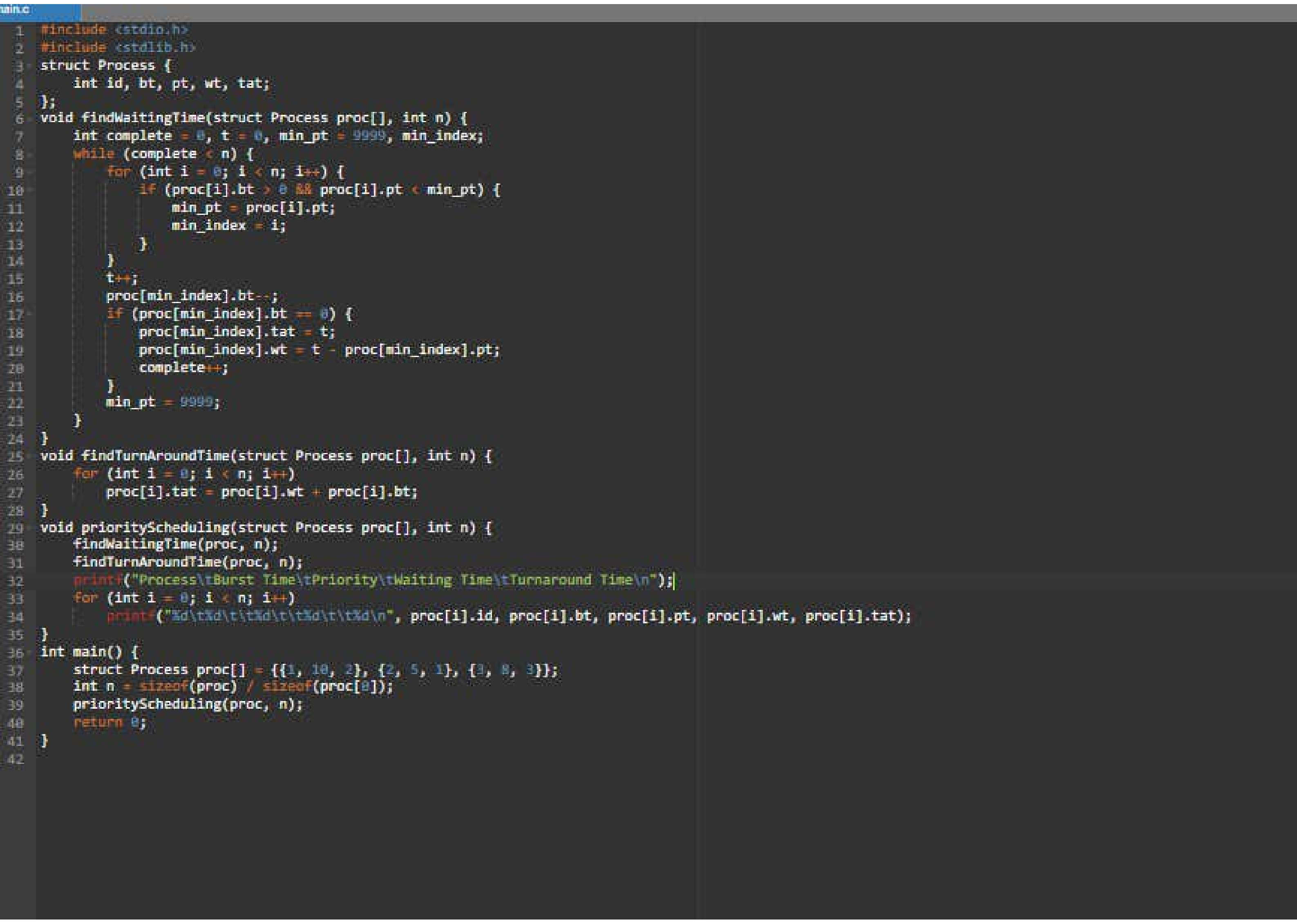
```c
#include <stdio.h>
#define MAX 10
typedef struct {
    int id;
    int priority;
} Process;

void schedule(Process p[], int n) {
    int highest = 0;
    for (int i = 1; i < n; i++) {
        if (p[i].priority > p[highest].priority) {
            highest = i;
        }
    }
    printf("Executing Process ID: %d with Priority: %d\n", p[highest].id, p[highest].priority);
}
int main() {
    Process p[MAX] = {{1, 2}, {2, 5}, {3, 1}, {4, 3}};
    schedule(p, 4);
    return 0;
}
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid, ppid;
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        ppid = getppid();
        printf("Child Process ID: %d\n", getpid());
        printf("parent Process ID: %d\n", ppid);
    } else {
        wait(NULL);
    }

    return 0;
}

```

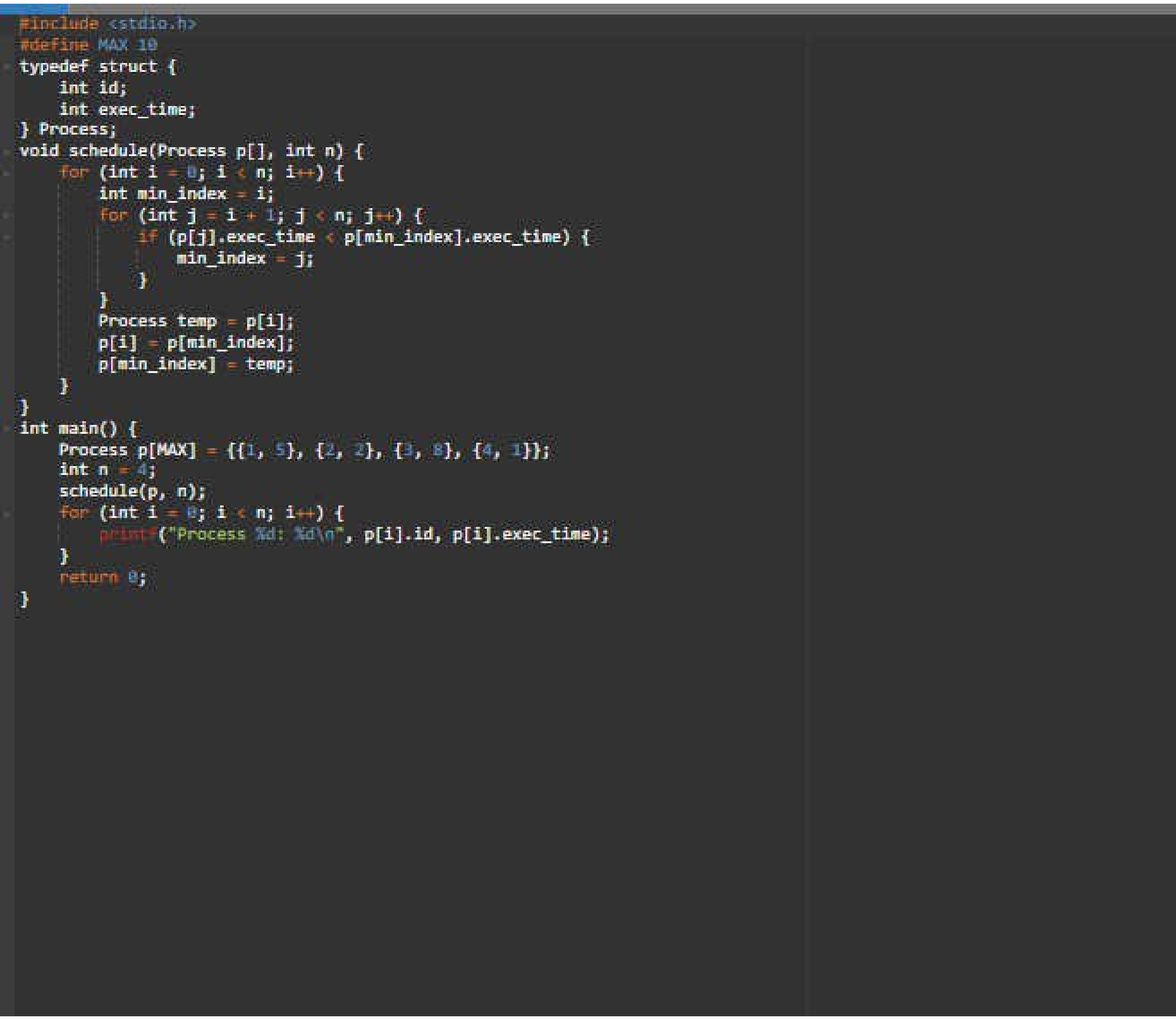


```c
#include <stdio.h>
#define MAX 10
typedef struct {
    int id;
    int exec_time;
} Process;
void schedule(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            if (p[j].exec_time < p[min_index].exec_time) {
                min_index = j;
            }
        }
        Process temp = p[i];
        p[i] = p[min_index];
        p[min_index] = temp;
    }
}
int main() {
    Process p[MAX] = {{1, 5}, {2, 2}, {3, 8}, {4, 1}};
    int n = 4;
    schedule(p, n);
    for (int i = 0; i < n; i++) {
        printf("Process %d: %d\n", p[i].id, p[i].exec_time);
    }
    return 0;
}
```

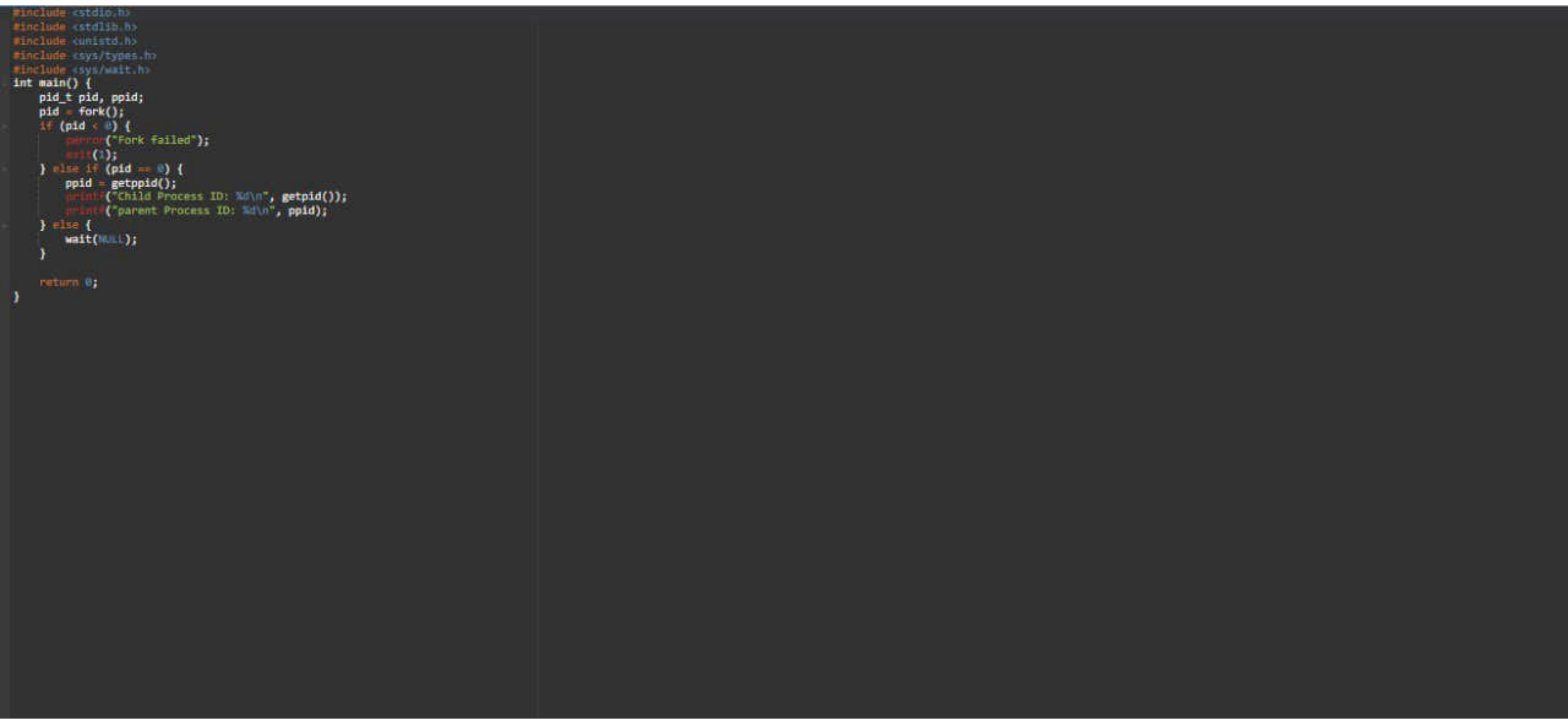


ain.c

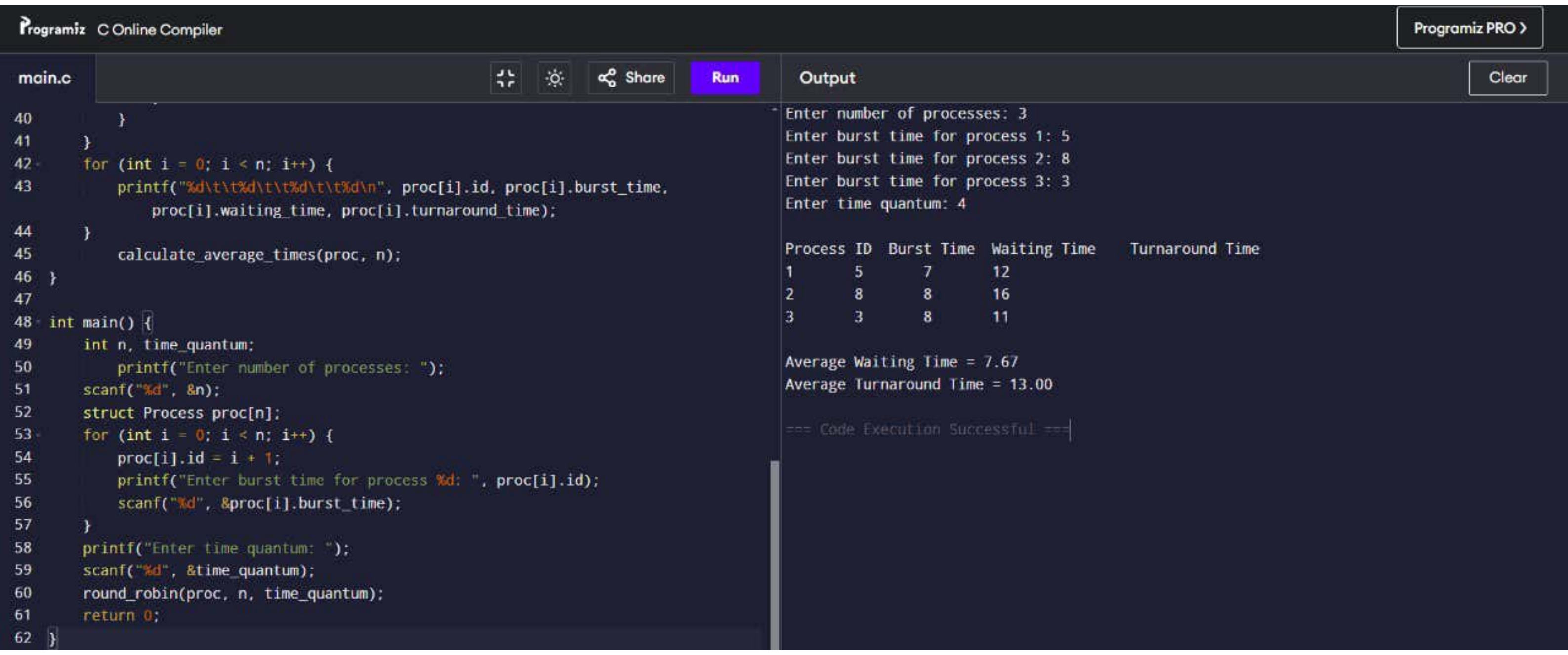
```c
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int id, bt, pt, wt, tat;
};
void findWaitingTime(struct Process proc[], int n) {
    int complete = 0, t = 0, min_pt = 9999, min_index;
    while (complete < n) {
        for (int i = 0; i < n; i++) {
            if (proc[i].bt > 0 && proc[i].pt < min_pt) {
                min_pt = proc[i].pt;
                min_index = i;
            }
        }
        t++;
        proc[min_index].bt--;
        if (proc[min_index].bt == 0) {
            proc[min_index].tat = t;
            proc[min_index].wt = t - proc[min_index].pt;
            complete++;
        }
        min_pt = 9999;
    }
}
void findTurnAroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++)
        proc[i].tat = proc[i].wt + proc[i].bt;
}
void priorityScheduling(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnAroundTime(proc, n);
    printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, proc[i].bt, proc[i].pt, proc[i].wt, proc[i].tat);
}
int main() {
    struct Process proc[] = {{1, 10, 2}, {2, 5, 1}, {3, 8, 3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    priorityScheduling(proc, n);
    return 0;
}
```



```c
#include <stdio.h>
#define MAX 10
typedef struct {
    int id;
    int exec_time;
} Process;
void schedule(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            if (p[j].exec_time < p[min_index].exec_time) {
                min_index = j;
            }
        }
        Process temp = p[i];
        p[i] = p[min_index];
        p[min_index] = temp;
    }
}
int main() {
    Process p[MAX] = {{1, 5}, {2, 2}, {3, 8}, {4, 1}};
    int n = 4;
    schedule(p, n);
    for (int i = 0; i < n; i++) {
        printf("Process %d: %d\n", p[i].id, p[i].exec_time);
    }
    return 0;
}
```

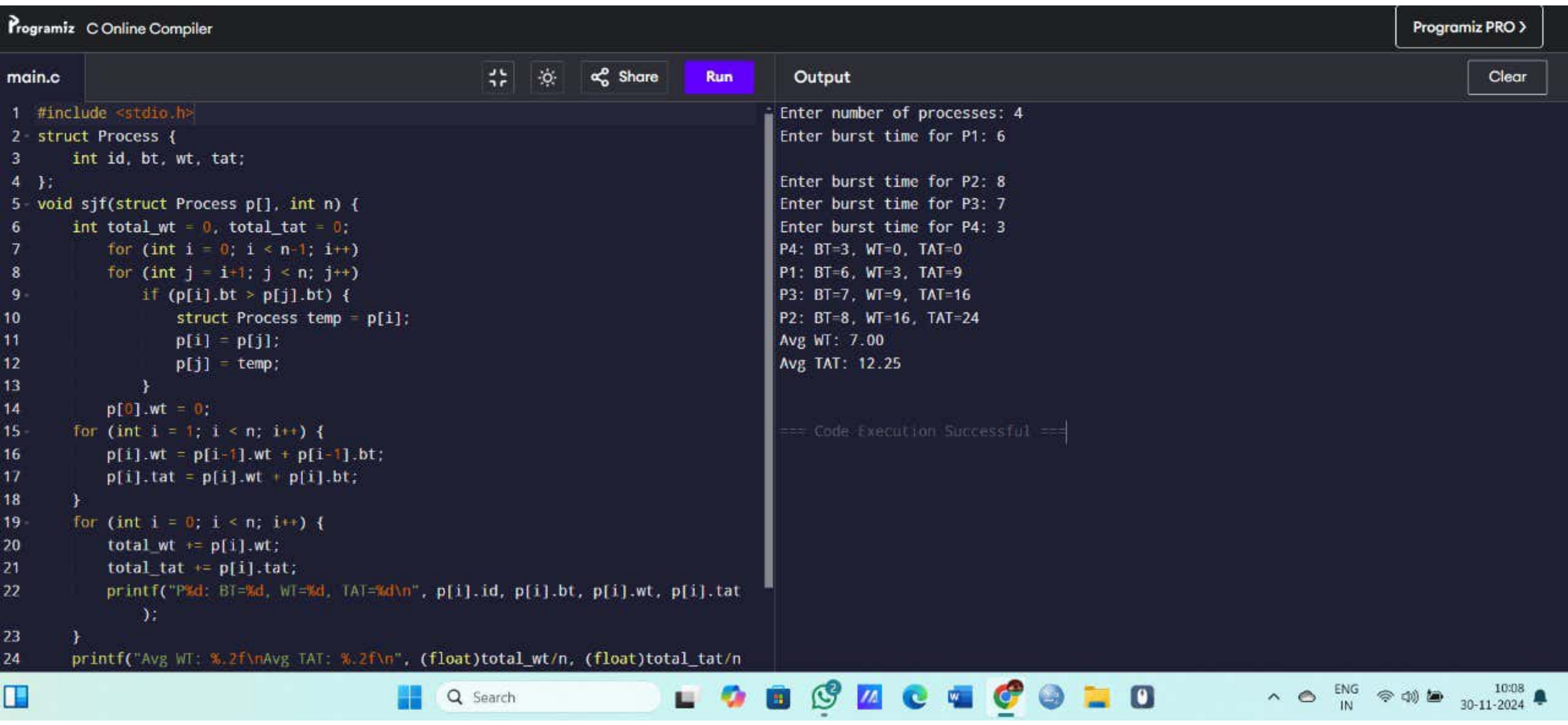


```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid, ppid;
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        ppid = getppid();
        printf("Child Process ID: %d\n", getpid());
        printf("parent Process ID: %d\n", ppid);
    } else {
        wait(NULL);
    }

    return 0;
}
```



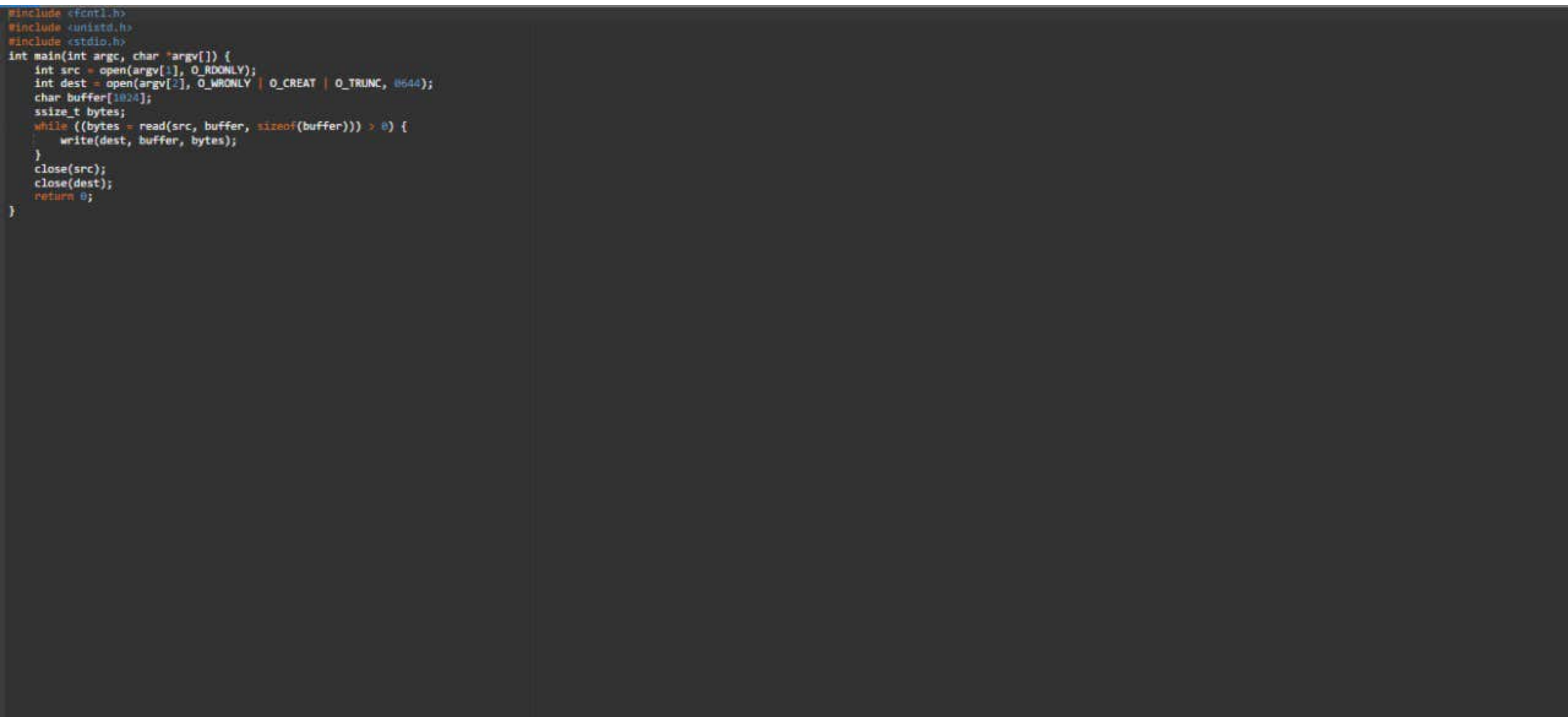
```c
        }
    }
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, proc[i].burst_time,
            proc[i].waiting_time, proc[i].turnaround_time);
    }
        calculate_average_times(proc, n);
}

int main() {
    int n, time_quantum;
        printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter burst time for process %d: ", proc[i].id);
        scanf("%d", &proc[i].burst_time);
    }
    printf("Enter time quantum: ");
    scanf("%d", &time_quantum);
    round_robin(proc, n, time_quantum);
    return 0;
}
```

```
Enter number of processes: 3
Enter burst time for process 1: 5
Enter burst time for process 2: 8
Enter burst time for process 3: 3
Enter time quantum: 4

Process ID  Burst Time  Waiting Time    Turnaround Time
1       5       7       12
2       8       8       16
3       3       8       11

Average Waiting Time = 7.67
Average Turnaround Time = 13.00

=== Code Execution Successful ===
```

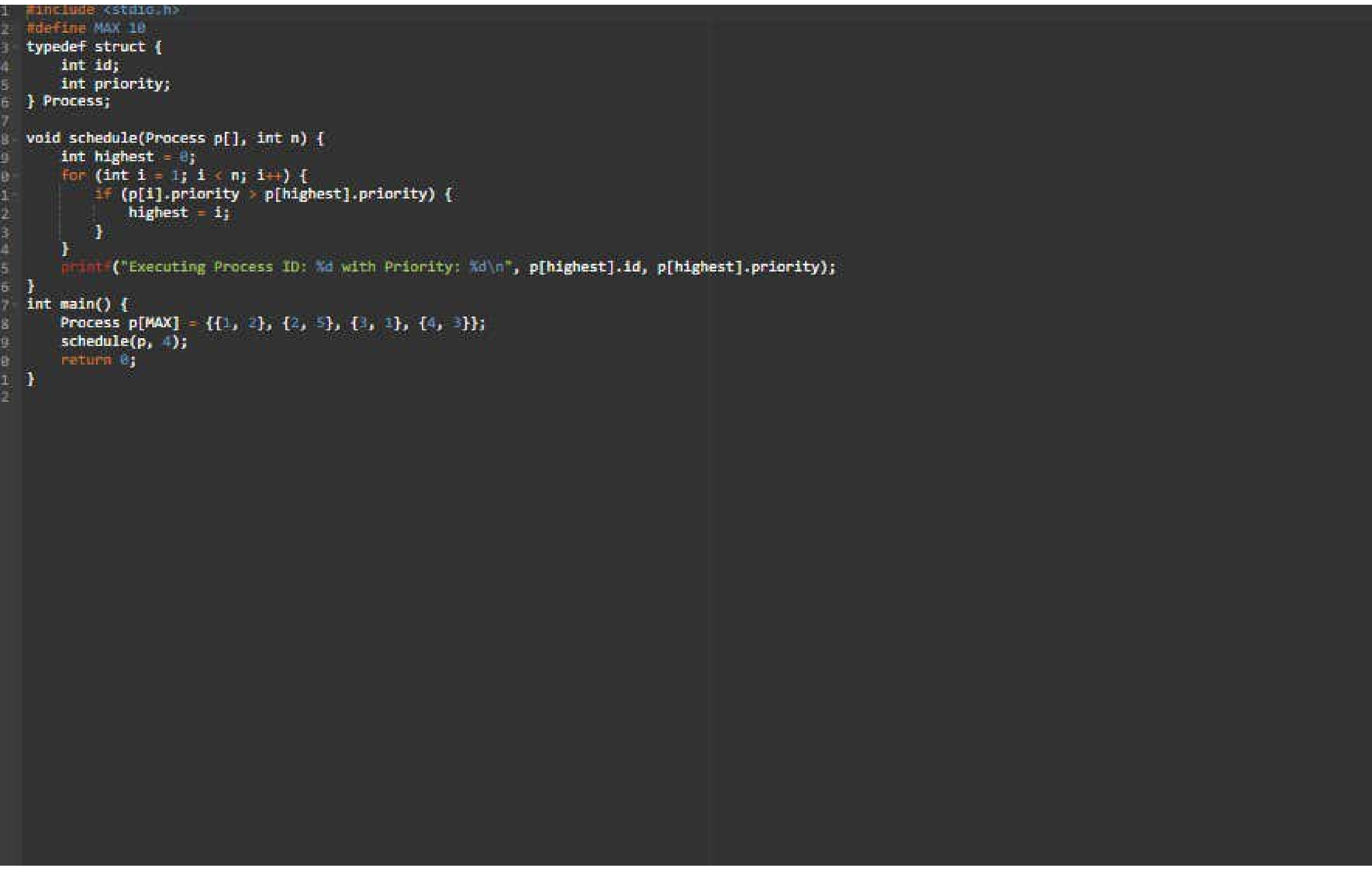


```c
#include <stdio.h>
struct Process {
    int id, bt, wt, tat;
};
void sjf(struct Process p[], int n) {
    int total_wt = 0, total_tat = 0;
        for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (p[i].bt > p[j].bt) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        p[0].wt = 0;
    for (int i = 1; i < n; i++) {
        p[i].wt = p[i-1].wt + p[i-1].bt;
        p[i].tat = p[i].wt + p[i].bt;
    }
    for (int i = 0; i < n; i++) {
        total_wt += p[i].wt;
        total_tat += p[i].tat;
        printf("P%d: BT=%d, WT=%d, TAT=%d\n", p[i].id, p[i].bt, p[i].wt, p[i].tat
            );
    }
    printf("Avg WT: %.2f\nAvg TAT: %.2f\n", (float)total_wt/n, (float)total_tat/n
```

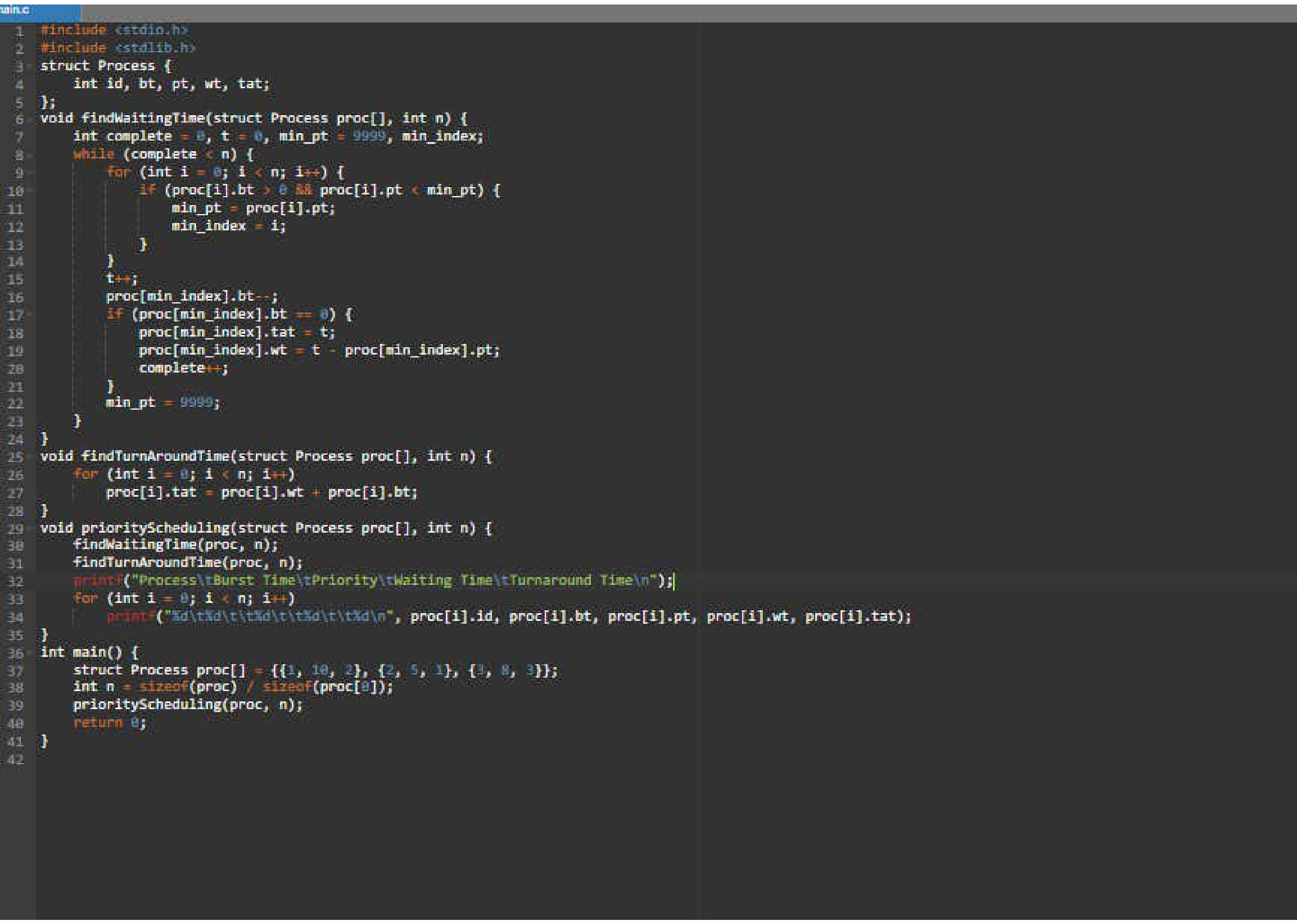
```
Enter number of processes: 4
Enter burst time for P1: 6

Enter burst time for P2: 8
Enter burst time for P3: 7
Enter burst time for P4: 3
P4: BT=3, WT=0, TAT=0
P1: BT=6, WT=3, TAT=9
P3: BT=7, WT=9, TAT=16
P2: BT=8, WT=16, TAT=24
Avg WT: 7.00
Avg TAT: 12.25


=== Code Execution Successful ===
```



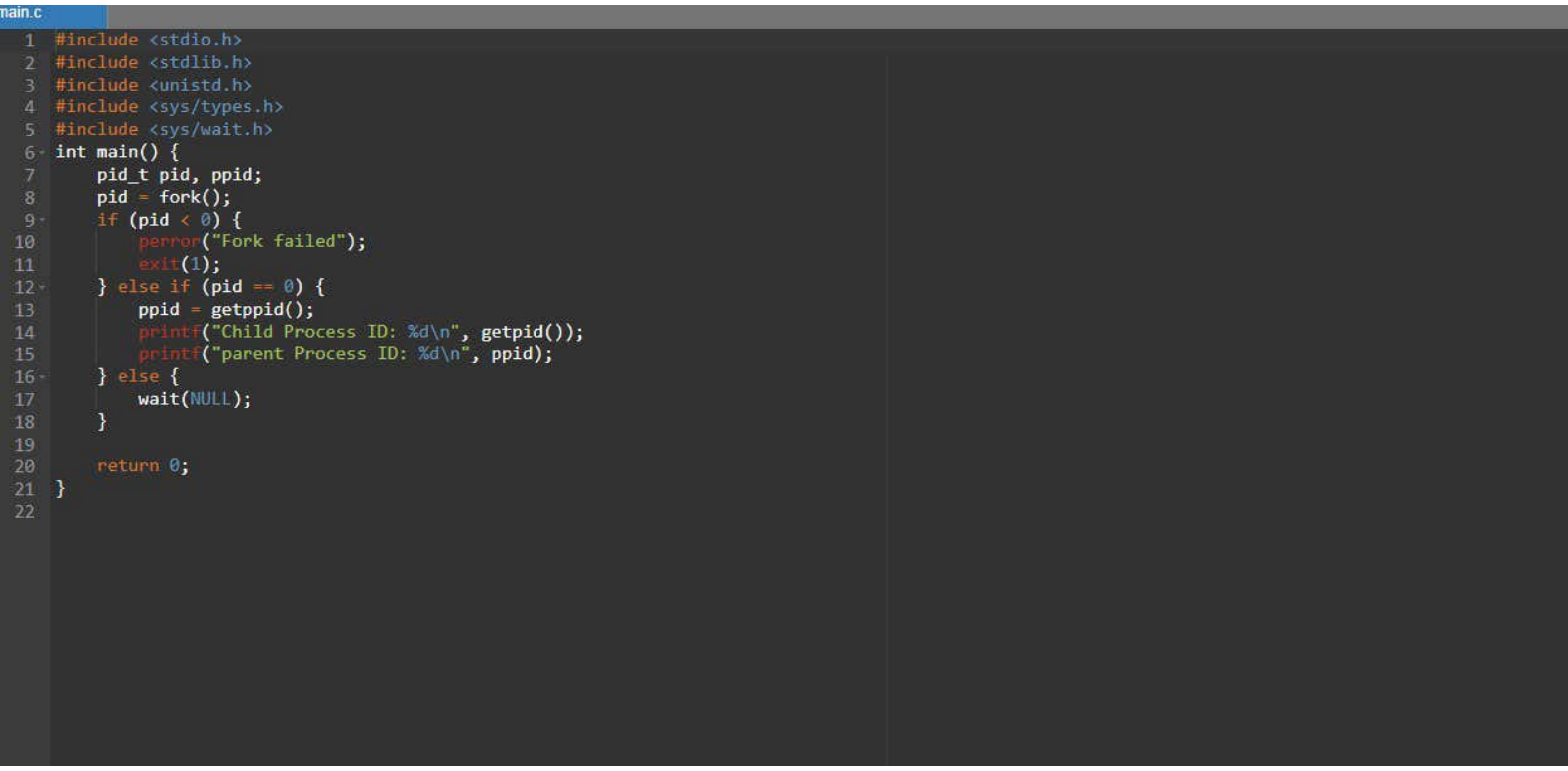
```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    int src = open(argv[1], O_RDONLY);
    int dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    char buffer[1024];
    ssize_t bytes;
    while ((bytes = read(src, buffer, sizeof(buffer))) > 0) {
        write(dest, buffer, bytes);
    }
    close(src);
    close(dest);
    return 0;
}
```



```c
#include <stdio.h>
#define MAX 10
typedef struct {
    int id;
    int priority;
} Process;

void schedule(Process p[], int n) {
    int highest = 0;
    for (int i = 1; i < n; i++) {
        if (p[i].priority > p[highest].priority) {
            highest = i;
        }
    }
    printf("Executing Process ID: %d with Priority: %d\n", p[highest].id, p[highest].priority);
}
int main() {
    Process p[MAX] = {{1, 2}, {2, 5}, {3, 1}, {4, 3}};
    schedule(p, 4);
    return 0;
}
```


main.c

```c
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int id, bt, pt, wt, tat;
};
void findWaitingTime(struct Process proc[], int n) {
    int complete = 0, t = 0, min_pt = 9999, min_index;
    while (complete < n) {
        for (int i = 0; i < n; i++) {
            if (proc[i].bt > 0 && proc[i].pt < min_pt) {
                min_pt = proc[i].pt;
                min_index = i;
            }
        }
        t++;
        proc[min_index].bt--;
        if (proc[min_index].bt == 0) {
            proc[min_index].tat = t;
            proc[min_index].wt = t - proc[min_index].pt;
            complete++;
        }
        min_pt = 9999;
    }
}
void findTurnAroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++)
        proc[i].tat = proc[i].wt + proc[i].bt;
}
void priorityScheduling(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnAroundTime(proc, n);
    printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, proc[i].bt, proc[i].pt, proc[i].wt, proc[i].tat);
}
int main() {
    struct Process proc[] = {{1, 10, 2}, {2, 5, 1}, {3, 8, 3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    priorityScheduling(proc, n);
    return 0;
}
```


main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid, ppid;
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        ppid = getppid();
        printf("Child Process ID: %d\n", getpid());
        printf("parent Process ID: %d\n", ppid);
    } else {
        wait(NULL);
    }

    return 0;
}
```